Students:
Nguyen Huu Truong Giang [u6110634@udg.edu]
Praveen Kumar Murali [u6110748@udg.edu]

Lab 4: Sample based path planners: RRT and RRT*

1 Introduction

Path planning is a fundamental problem in robotics, where the goal is to find a valid route from a starting point to a destination without hitting obstacles. Traditional methods can be slow or difficult to use in complex, high-dimensional spaces. To solve this, we use Sampling-based Motion Planners (SBMP). These algorithms do not build a full map of the world; instead, they explore the space by taking random samples.

In this lab, we implemented two specific algorithms in Python: the Rapidly-exploring Random Tree (RRT) and its optimized version, RRT*. RRT is effective for quickly exploring free space. RRT* adds an optimization step, meaning that as the number of nodes increases (approaches infinity), the path converges to the shortest possible solution. We tested these algorithms on 2D grid maps derived from grayscale images and implemented a smoothing step to improve the final path quality.

2 Methodology

The implementation was done in Python using the PIL library to load maps and Matplotlib to visualize the trees. The specific steps for the environment and algorithms are described below.

2.1 Environment Setup

We used grayscale images to represent the environment.

1. The image is loaded and converted into a 2D numpy array.
2. The array is binarized to ensure values are only 0 or 1.
3. Standard images use 255 (white) for empty space, but we needed 0 to be free space and 1 to be an obstacle. Therefore, we inverted the values.

2.2 RRT Algorithm

The RRT algorithm builds a tree structure T to connect q_{start} to q_{goal}. We followed the logic outlined in Algorithm 1 [1].

Algorithm 1 RRT Algorithm

```
1: **Input:** $C$ (config space), $K$ (iterations), $\Delta t$ (step), $p$ (goal bias), $q_{start}, q_{goal}$
2: **Output:** Tree $T = (V, E)$ and path (if found)
3: $V \leftarrow \{q_{start}\}, \quad E \leftarrow \emptyset$
4: for $k = 1$ to $K$ do
5:     $q_{rand} \leftarrow$ RAND_CONF$(C, p, q_{goal})$
6:     $q_{near} \leftarrow$ NEAREST$(V, q_{rand})$
7:     $q_{new} \leftarrow$ STEER$(q_{near}, q_{rand}, \Delta t)$
8:     if COLLISION_FREE$(q_{near}, q_{new})$ then
9:         $V \leftarrow V \cup \{q_{new}\}; \quad E \leftarrow E \cup \{(q_{near}, q_{new})\}$
10:        if GOAL_REACHED$(q_{new}, q_{goal})$ then
11:            return $(T,$ EXTRACT_PATH$(T, q_{start}, q_{goal}))$
12:        end if
13:    end if
14: end for
15: return $(T,$ "No solution found"$)$
```

The process works as follows:

- **Sampling:** A random point q_{rand} is generated. With probability p, we pick the goal position to guide the tree; otherwise, it is random.
- **Nearest Neighbor:** We find the closest existing node q_{near} in the tree.
- **Steering:** A new node q_{new} is created by moving distance Δt from q_{near} towards q_{rand}.
- **Collision Check:** We check if the line between the nodes is free of obstacles using an incremental strategy.

2.3 Collision Free

For collision checking, there are two common approaches to determine whether a line segment intersects any obstacles: the incremental method and the bisection method, illustrated in Figure 1. In our implementation, we choose the bisection method due to its computational efficiency. The pseudocode for this method is provided in Algorithm 2.

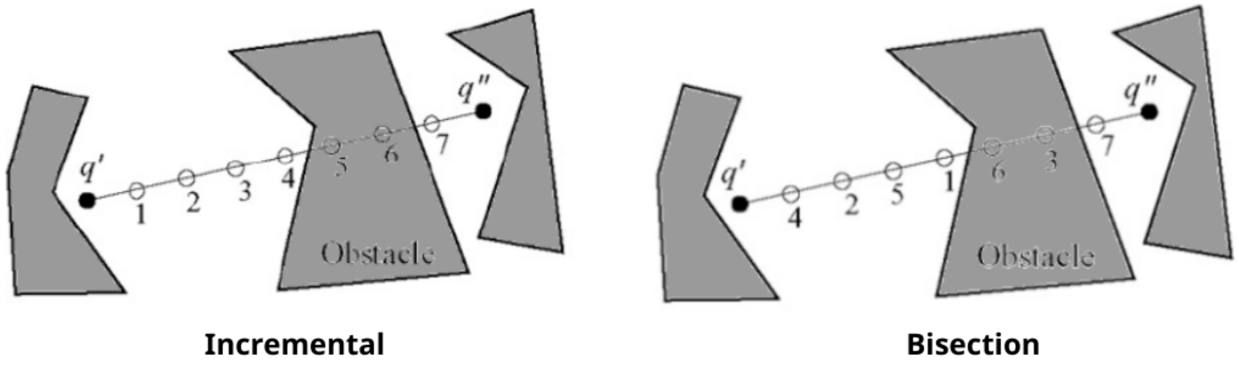


Figure 1: Methods to check collision free [2]

Algorithm 2 IS_FREE_SEGMENT($q_{\text{near}}, q_{\text{new}}, C, \text{depth}, \text{max\_depth}$)

```
if $\text{depth} \geq \text{max\_depth}$ then
    return True
end if
if POINT_COLLIDED($q_{\text{new}}, C$) then
    return False
end if
$q_{\text{mid}} \leftarrow (q_{\text{near}} + q_{\text{new}})/2$
if POINT_COLLIDED($q_{\text{mid}}, C$) then
    return False
end if
$left \leftarrow$ IS_FREE_SEGMENT($q_{\text{near}}, q_{\text{mid}}, C, \text{depth}+1, \text{max\_depth}$)
if $left =$ False then
    return False
end if
$right \leftarrow$ IS_FREE_SEGMENT($q_{\text{mid}}, q_{\text{new}}, C, \text{depth}+1, \text{max\_depth}$)
return $right$
```

2.4 Path Smoothing

The path found by RRT is often jagged. We implemented a greedy smoothing algorithm to fix this. We attempt to connect the start node directly to the goal node. If that line hits an obstacle, we try connecting to the previous node in the path, repeating this until a valid straight-line shortcut is found.

2.5 RRT* Algorithm

RRT* improves upon RRT by optimizing the path cost. It introduces two extra steps when adding a node, as shown in Algorithm 3 [1].

Algorithm 3 RRT* Main Algorithm

```
1: Input: $C, K, \Delta t, p, q_{start}, q_{goal}$
2: Output: Tree $T$ and path
3: $V \leftarrow \{q_{start}\}, E \leftarrow \emptyset; \quad J[q_{start}] \leftarrow 0$
4: for $k = 1$ to $K$ do
5:     $q_{rand} \leftarrow$ RAND_CONF$(C, p, q_{goal})$
6:     $q_{near} \leftarrow$ NEAREST$(V, q_{rand})$
7:     $q_{new} \leftarrow$ STEER$(q_{near}, q_{rand}, \Delta t)$
8:     if COLLISION_FREE$(q_{rand}, q_{new}, C)$ then
9:         $T \leftarrow$ REWIRE_QNEAR_TO_QNEW$(T, q_{near}, q_{new}, J)$
10:        $T \leftarrow$ REWIRE_QNEW_FROM_QNEAR$(T, q_{new}, J)$
11:        if $q_{new} == q_{goal}$ then
12:            return $(T,$ EXTRACT_PATH$(T, q_{start}, q_{goal}))$
13:        end if
14:    end if
15: end for
16: return $(T,$ "No solution found")
```

The optimizations are:

1. **Cost Optimization:** We connect q_{new} to the neighbor that gives the lowest total path cost from the start, not just the nearest one.

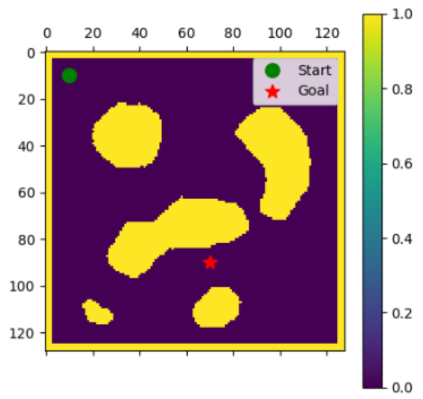
2. **Rewiring:** After adding q_{new}, we check if any neighbors can reach the start faster by going through q_{new}. If so, we change their parent to q_{new}.

3 Results

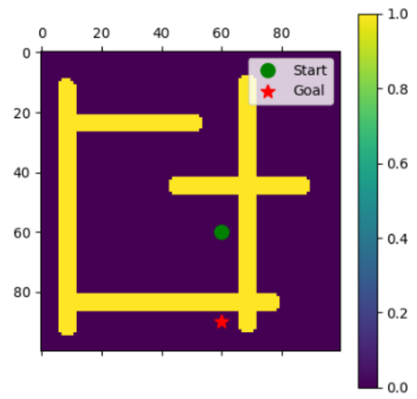
We conducted experiments using the **RRT and RRT* algorithm** on four different maps (Map 0, Map 1, Map 2, and Map 3), as illustrated in Figure 2. In each map, the **green circle** represents the start position, while the **red star** indicates the goal position. The position configurations for each experiment are summarized in Table 1.

Map	Start	Goal
0	$[10, 10]$	$[90, 70]$
1	$[60, 60]$	$[90, 60]$
2	$[8, 31]$	$[139, 38]$
3	$[50, 90]$	$[375, 375]$

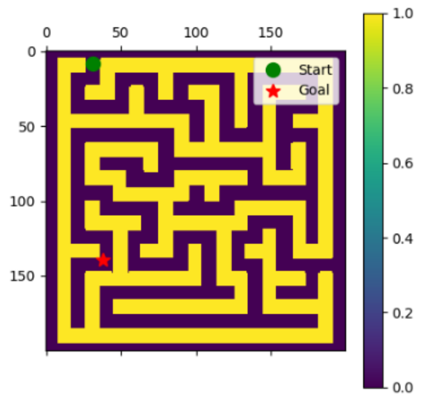
Table 1: Start and Goal configuration for each map



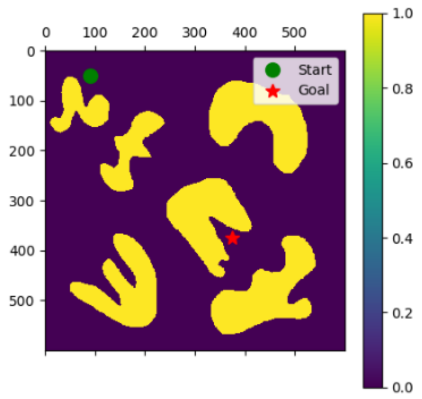
Map 0



Map 1



Map 2



Map 3

Figure 2: Map 0, 1, 2, and 3 with start and goal position

3.1 RRT

To implement the **RRT algorithm**, we must define several hyperparameters for each map so that the method can successfully generate a path in environments of varying complexity. The hyperparameters selected for different map configurations are listed in Table 2.

Map	K	Δq	p
0	1000	20	0.2
1	2000	15	0.2
2	20000	10	0.2
3	1000	25	0.2

Table 2: Hyper-parameters configuration for **RRT algorithm** for each map

After setting the hyperparameters, we conducted our experiments on four different maps, both with and without path smoothing, to evaluate the performance of the **RRT algorithm** in environments of varying complexity. The visualization of the results is shown in Figure 3, and the quantitative comparison is provided in Table 3. The table reports the total path cost (with and without smoothing), demonstrating the effectiveness of the smoothing step, as well as the number of iterations, which highlights how map complexity influences the convergence of the algorithm. Moreover, we clearly observe that the smoothed path cost is significantly lower than the original path cost, confirming the benefit of applying a post-processing smoothing step.

Map	Iterations	Total path cost	Total smooth path cost
0	37	152.34	134.16
1	772	294.27	229.88
2	13643	962.36	886.80
3	245	599.57	511.46

Table 3: Iterations and total path cost with and without smoothing using RRT algorithm

3.2 RRT Star

Unlike the **RRT algorithm**, where the path found is generally not optimal, the **RRT*-algorithm** progressively improves the solution and converges toward a near-optimal path as the number of sampled nodes increases. To implement **RRT***, we also define a set of hyperparameters for each map, as listed in Table 4.

Map	K	Δq	p	Max Range
0	1000	5	0.2	30
1	3000	10	0.2	20
2	20000	10	0.2	30
3	4000	15	0.2	40

Table 4: Hyper-parameters configuration for **RRT-Star algorithm** for each map

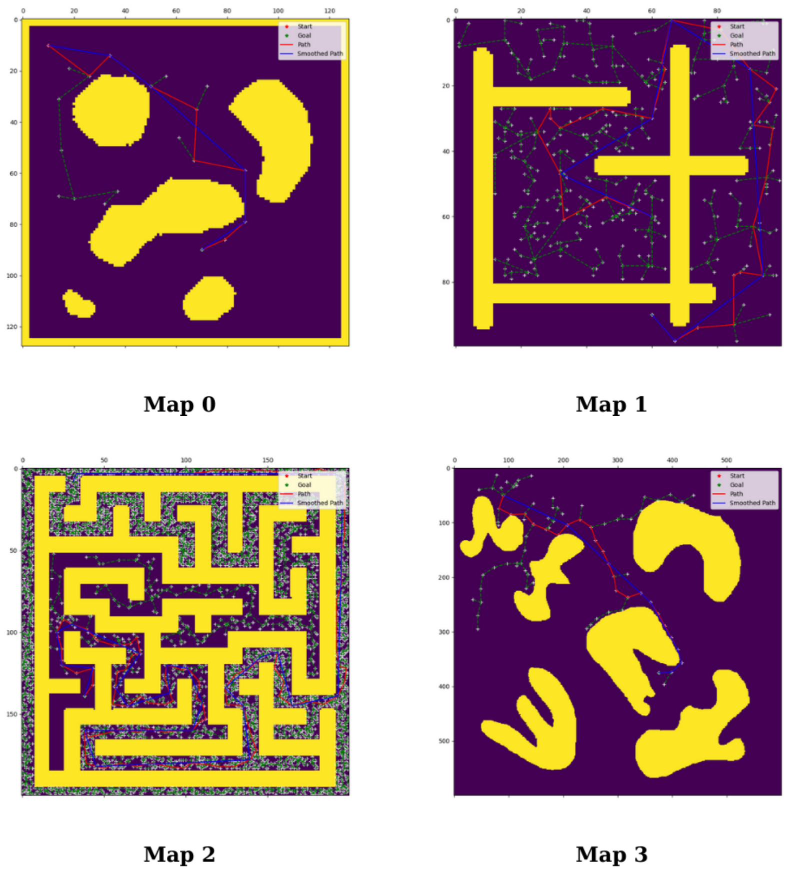
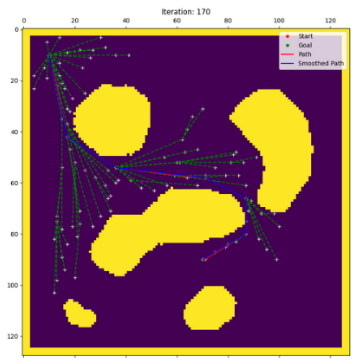


Figure 3: Path from start to goal position using RRT algorithm with and without smoothing method in different maps

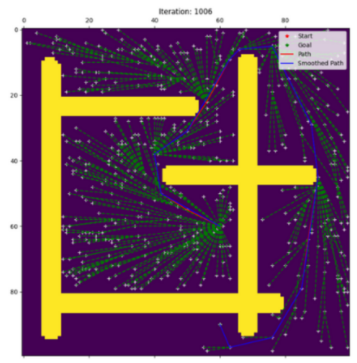
The results in Table 5 show that the total smoothed path cost decreases as the number of sampled nodes increases. This improvement can also be observed visually in Figures 4 and 5, where the path progressively converges toward an optimal solution.

Map	Iterations	Total path cost first found	Iterations	Total path cost converge
0	170	142.07	658	133.80
1	1006	200.58	1115	198.91
2	10506	593.12	14084	578.50
3	161	608.04	4000	518.73

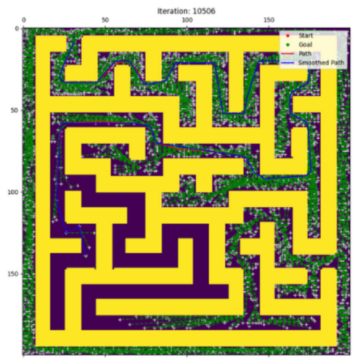
Table 5: Iterations and total path cost with and without smoothing using RRT star algorithm



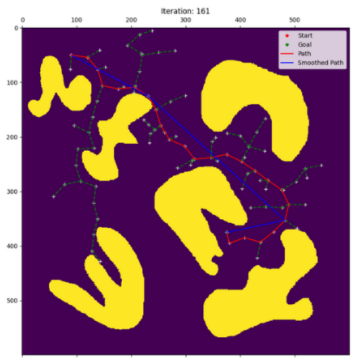
Map 0



Map 1



Map 2



Map 3

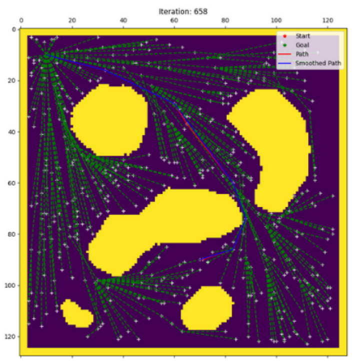
Figure 4: Path first time found from start to goal position using RRT-star algorithm with and without smoothing in different maps

4 Discussions and Conclusions

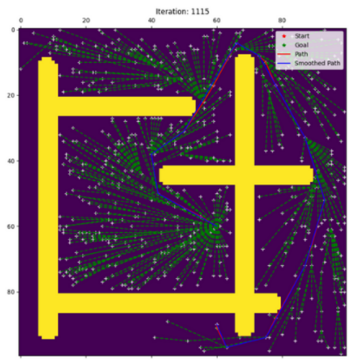
Nowadays, the **RRT and RRT* algorithms** are widely used in path-planning problems due to their effectiveness in high–degree-of-freedom task spaces, where traditional graph-search methods become computationally expensive. However, implementing these algorithms requires selecting several hyperparameters, which must be tuned for each specific scenario. Moreover, **RRT and RRT* do not inherently guarantee optimal solutions**; RRT* converges toward an optimal path only as the number of samples approaches infinity.

Another limitation is that both algorithms perform poorly in environments with narrow passages, where random sampling seldom reaches feasible regions. As observed in our experiments, particularly in map 2, this significantly increases the number of iterations required to find a valid path.

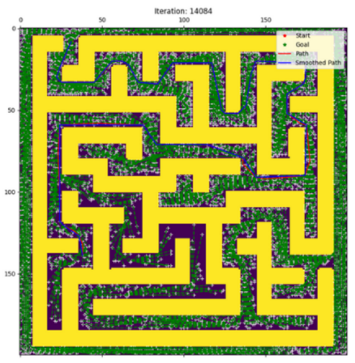
In summary, the **RRT and RRT* algorithms** remain powerful and widely used approaches for path planning, and they continue to be the focus of extensive research. Many improvements



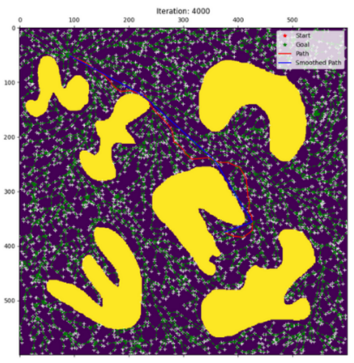
Map 0



Map 1



Map 2



Map 3

Figure 5: Path converge found from start to goal position using RRT-star algorithm with and without smoothing in different maps

and variants have been proposed to address the limitations discussed above, further enhancing their efficiency, optimality, and applicability in complex environments.

References

[1] N. Palomeras, "Slide autonomous systems sampling based motion planning," *Universitat de Girona*. `https://moodle.udg.edu/pluginfile.php/2491459/mod_resource/content/3/sampling_base.pdf`.

[2] N. Palomeras, "Lab document for rrt and rrt star," *Universitat de Girona*. `https://moodle.udg.edu/course/section.php?id=371461#module-1784423`.